

C API

```
#ifdef __cplusplus
/*
 * We put all the zoidfs stuff in the zoidfs namespace
 */
namespace zoidfs
{
    extern "C"
    {
#endif

/***
 * Handle is an opaque 32 byte value
 *
 * NOTE for implementors: the first 4 bytes are reserved
 * and should be ignored by the implementation.
 */
typedef struct
{
    uint8_t data[32];
} zoidfs_handle_t;

typedef enum
{
    ZOIDFS_INVAL = 0,
    ZOIDFS_REG = 1,
    ZOIDFS_DIR = 2,
    ZOIDFS_LNK = 3,
    ZOIDFS_CHR = 4,
    ZOIDFS_BLK = 5,
    ZOIDFS_FIFO = 6,
    ZOIDFS SOCK = 7
} zoidfs_attr_type_t;

#define ZOIDFS_ATTR_MODE (1 << 0)
#define ZOIDFS_ATTR_NLINK (1 << 1)
#define ZOIDFS_ATTR_UID (1 << 2)
#define ZOIDFS_ATTR_GID (1 << 3)
#define ZOIDFS_ATTR_SIZE (1 << 4)
#define ZOIDFS_ATTR_BSIZE (1 << 5)
#define ZOIDFS_ATTR_FSID (1 << 6)
#define ZOIDFS_ATTR_FILEID (1 << 7)
#define ZOIDFS_ATTR_ATIME (1 << 8)
#define ZOIDFS_ATTR_MTIME (1 << 9)
#define ZOIDFS_ATTR_CTIME (1 << 10)

/* NOTE: the following is only used in the 'flags' field of zoidfs_readdir */
#define ZOIDFS_RETR_HANDLE (1 << 16)

#define ZOIDFS_ATTR_SETABLE (ZOIDFS_ATTR_MODE | \
                           ZOIDFS_ATTR_UID | \
                           ZOIDFS_ATTR_GID | \
                           ZOIDFS_ATTR_ATIME | \
                           ZOIDFS_ATTR_MTIME)

#define ZOIDFS_ATTR_SETTABLE ZOIDFS_ATTR_SETABLE
```

```

#define ZOIDFS_ATTR_ALL (ZOIDFS_ATTR_MODE | \
    ZOIDFS_ATTR_NLINK | \
    ZOIDFS_ATTR_UID | \
    ZOIDFS_ATTR_GID | \
    ZOIDFS_ATTR_SIZE | \
    ZOIDFS_ATTR_BSIZE | \
    ZOIDFS_ATTR_FSID | \
    ZOIDFS_ATTR_FILEID | \
    ZOIDFS_ATTR_ATIME | \
    ZOIDFS_ATTR_MTIME | \
    ZOIDFS_ATTR_CTIME)

/**
 * NOTE: follows unix convention: seconds since 1/1/1970
 */
typedef struct
{
    uint64_t seconds;
    uint32_t nseconds;
} zoidfs_time_t;

/**
 * NOTE: mode uses the POSIX file permission bits
 */
typedef struct
{
    zoidfs_attr_type_t type;      uint16_t mask;
    uint16_t mode;
    uint32_t nlink;
    uint32_t uid;
    uint32_t gid;
    uint64_t size;
    uint32_t blocksize;
    uint32_t fsid;
    uint64_t fileid;
    zoidfs_time_t atime;
    zoidfs_time_t mtime;
    zoidfs_time_t ctime;
} zoidfs_attr_t;

typedef struct
{
    uint64_t size;
    zoidfs_time_t atime;
    zoidfs_time_t mtime;
    zoidfs_time_t ctime;
} zoidfs_cache_hint_t;

typedef struct
{
    uint16_t mask;
    uint16_t mode;
    uint32_t uid;
    uint32_t gid;
    uint64_t size;
    zoidfs_time_t atime;
    zoidfs_time_t mtime;
} zoidfs_sattr_t;

/* The cookie is an opaque object */
typedef uint64_t zoidfs_dirent_cookie_t;

```

```

/* Cookie entry for the first directory entry */
#define ZOIDFS_DIRENT_COOKIE_INIT ((uint64_t) 0)

typedef struct
{
    char name[ZOIDFS_NAME_MAX+1];
    zoidfs_handle_t handle;
    zoidfs_attr_t attr;
    zoidfs_dirent_cookie_t cookie;
} zoidfs_dirent_t;

/***
 * XXX: These need to be sorted by severity
 */
enum {
    ZFS_OK = 0,
    ZFSERR_PERM=1,
    ZFSERR_NOENT=2,
    ZFSERR_IO=5,
    ZFSERR_NXIO=6,
    ZFSERR_NOMEM=12,
    ZFSERR_ACES=13,
    ZFSERR_EXIST=17,
    ZFSERR_NODEV=19,
    ZFSERR_NOTDIR=20,
    ZFSERR_ISDIR=21,
    ZFSERR_INVAL=22,
    ZFSERR_FBIG=27,
    ZFSERR_NOSPC=28,
    ZFSERR_ROFS=30,
    ZFSERR_NOTIMPL=38,
    ZFSERR_NAMETOOLONG=63,
    ZFSERR_NOTEEMPTY=66,
    ZFSERR_DQUOT=69,
    ZFSERR_STALE=70,
    ZFSERR_WFLUSH=99,
    ZFSERR_OTHER=99999
};

/***
 * OPTIONAL
 */
int zoidfs_init(void);

/***
 * OPTIONAL
 */
int zoidfs_finalize(void);

/* START-ZOID-SCANNER ID=1 INIT=__zoidfs_init FINI=__zoidfs_finalize PROC=NULL */

int zoidfs_null(void);

/***
 * NOTE:
 * - setattr acts as lstat, i.e. it does not follow symbolic links
 * - attr.mask indicates which values need to be retrieved.
 * - Other fields are ignored on input. */
int zoidfs_getattr(const zoidfs_handle_t * handle /* in:ptr */,
                   zoidfs_attr_t * attr /* inout:ptr */);

```

zoidfs_setattr

```
int zoidfs_setattr(const zoidfs_handle_t * handle /* in:ptr */,
                   const zoidfs_sattr_t * sattr /* in:ptr */,
                   zoidfs_attr_t * attr /* inout:ptr:nullok */);
```

This function can set attributes and optionally retrieve them.

- On input, only the mask field of attr is relevant.
- If sattr.mode and attr.mode contain shared bits, attr will retrieve the modified values.
- attr->mask is used to determine which, if any, attributes need to be retrieved.

zoidfs_lookup

```
int zoidfs_lookup(const zoidfs_handle_t * parent_handle /* in:ptr:nullok */,
                  const char * component_name /* in:str:nullok */,
                  const char * full_path /* in:str:nullok */,
                  zoidfs_handle_t * handle /* out:ptr */);
```

- Lookup follows POSIX conventions: if the directory is accessible, even though the file is not, a valid zoidfs handle will be returned (and the handle can be used in zoidfs_getattr).
- Lookup can return ESTALE. In this case, the user is responsible for performing a lookup on the parent directory until a valid handle is returned.
- Note that full path should be absolute (and start with /).

```
int zoidfs_readlink(const zoidfs_handle_t * handle /* in:ptr */,
                     char * buffer /* out:arr:size+=1 */,
                     size_t buffer_length /* in:obj */);
```

Read and Write

```
int zoidfs_read(const zoidfs_handle_t * handle /* in:ptr */,
                size_t mem_count /* in:obj */,
                void * mem_starts[] /* out:arr2d:size+=1:zerocopy */,
                const size_t mem_sizes[] /* in:arr:size=-2 */,
                size_t file_count /* in:obj */,
                const uint64_t file_starts[] /* in:arr:size=-1 */,
                uint64_t file_sizes[] /* inout:arr:size=-2 */);

int zoidfs_write(const zoidfs_handle_t * handle /* in:ptr */,
                 size_t mem_count /* in:obj */,
                 const void * mem_starts[] /* in:arr2d:size+=1:zerocopy */,
                 const size_t mem_sizes[] /* in:arr:size=-2 */,
                 size_t file_count /* in:obj */,
                 const uint64_t file_starts[] /* in:arr:size=-1 */,
                 uint64_t file_sizes[] /* inout:arr:size=-2 */);
```

* Is there a limitation on mem_count or file_count?

* Do offsets need to be increasing in memory? in file?

Commit

```
int zoidfs_commit(const zoidfs_handle_t * handle /* in:ptr */);

/***
 * NOTE: if the file already exists zoidfs_create will lookup the handle
 * and return success but set created to 0
 */
int zoidfs_create(const zoidfs_handle_t * parent_handle /* in:ptr:nullok */,
                  const char * component_name /* in:str:nullok */,
                  const char * full_path /* in:str:nullok */,
                  const zoidfs_sattr_t * attr /* in:ptr */,
                  zoidfs_handle_t * handle /* out:ptr */,
                  int * created /* out:ptr:nullok */);

int zoidfs_remove(const zoidfs_handle_t * parent_handle /* in:ptr:nullok */,
                  const char * component_name /* in:str:nullok */,
                  const char * full_path /* in:str:nullok */,
                  zoidfs_cache_hint_t * parent_hint /* out:ptr:nullok */);

int zoidfs_rename(const zoidfs_handle_t * from_parent_handle /* in:ptr:nullok */,
                  const char * from_component_name /* in:str:nullok */,
                  const char * from_full_path /* in:str:nullok */,
                  const zoidfs_handle_t * to_parent_handle /* in:ptr:nullok */,
                  const char * to_component_name /* in:str:nullok */,
                  const char * to_full_path /* in:str:nullok */,
                  zoidfs_cache_hint_t * from_parent_hint /* out:ptr:nullok */,
                  zoidfs_cache_hint_t * to_parent_hint /* out:ptr:nullok */);
```

Link and Symlink

```
int zoidfs_link(const zoidfs_handle_t * from_parent_handle /* in:ptr:nullok */,
                 const char * from_component_name /* in:str:nullok */,
                 const char * from_full_path /* in:str:nullok */,
                 const zoidfs_handle_t * to_parent_handle /* in:ptr:nullok */,
                 const char * to_component_name /* in:str:nullok */,
                 const char * to_full_path /* in:str:nullok */,
                 zoidfs_cache_hint_t * from_parent_hint /* out:ptr:nullok */,
                 zoidfs_cache_hint_t * to_parent_hint /* out:ptr:nullok */);

int zoidfs_symlink(const zoidfs_handle_t * from_parent_handle /* in:ptr:nullok */,
                   const char * from_component_name /* in:str:nullok */,
                   const char * from_full_path /* in:str:nullok */,
                   const zoidfs_handle_t * to_parent_handle /* in:ptr:nullok */,
                   const char * to_component_name /* in:str:nullok */,
                   const char * to_full_path /* in:str:nullok */,
                   const zoidfs_sattr_t * attr /* in:ptr */,
                   zoidfs_cache_hint_t * from_parent_hint /* out:ptr:nullok */,
                   zoidfs_cache_hint_t * to_parent_hint /* out:ptr:nullok */);
```

* In both cases, **from** refers to the **already existing file**, while **to** refers to the newly created entry. * It is considered an error if the new entry already exists.

mkdir

```
int zoidfs_mkdir(const zoidfs_handle_t * parent_handle /* in:ptr:nullok */,
                  const char * component_name /* in:str:nullok */,
                  const char * full_path /* in:str:nullok */,
                  const zoidfs_sattr_t * attr /* in:ptr */,
                  zoidfs_cache_hint_t * parent_hint /* out:ptr:nullok */);

/***
 * NOTE:
 *   const zoidfs_sattr_t * attr /* in:ptr */,
 *   zoidfs_cache_hint_t * from_parent_hint /* out:ptr:nullok */,
 *   zoidfs_cache_hint_t * to_parent_hint /* out:ptr:nullok */);

int zoidfs_mkdir(const zoidfs_handle_t * parent_handle /* in:ptr:nullok */,
                  const char * component_name /* in:str:nullok */,
                  const char * full_path /* in:str:nullok */,
                  const zoidfs_sattr_t * attr /* in:ptr */,
                  zoidfs_cache_hint_t * parent_hint /* out:ptr:nullok */);

/***
 * NOTE:
 *   - zoidfs_readdir return . and ..
 *   (if this causes problems we might reconsider)
 *   - the flags field indicates which attributes need to be retrieved
 *   - In addition, a special flag ZOIDFS_RETR_HANDLE is accepted
 *   and causes readdir to return the zoidfs handle of each directory
 *   entry.
 */
int zoidfs_readdir(const zoidfs_handle_t * parent_handle /* in:ptr */,
                   zoidfs_dirent_cookie_t cookie /* in:obj */,
                   size_t * entry_count /* inout:ptr */,
                   zoidfs_dirent_t * entries /* out:arr:size=-1 */,
                   uint32_t flags /* in:obj */,
                   zoidfs_cache_hint_t * parent_hint /* out:ptr:nullok */);

int zoidfs_resize(const zoidfs_handle_t * handle /* in:ptr */,
                  uint64_t size /* in:obj */);

#endif __cplusplus
} /* extern C */
} /* namespace */
#endif
```

Tricky Points

- * What do we do when the user does a lookup on a symlink?
- * How do we handle deleting a file that is in use by others (i.e. others have a handle for it). This might differ depending on the dispatcher backend
- * Is a handle connected to a filename or to the file contents? If we rename a file, does the handle change?

* Do two hardlinked files have the same zoidfs handle?

* Does a lookup fail on a file if the file exists but we don't have any permission to access the file?

Follow POSIX here: if the directory is readable, a lookup (and getattr) on a file in that directory is allowed.

* Does the lookup fail if we don't have directory permission for a directory along the path?

* Do we assume that the filename -> handle mapping is secret?